[Power BI Data Modeling] [cheatsheet]

1. Table Creation

- Create a new table: `Table = Table.FromRows({{1, "John"}, {2, "Jane"}}, {"ID", "Name"})`
- Create a table from a CSV file: `Table = Csv.Document(File.Contents("C:\data.csv"))`
- Create a table from an Excel file: `Table = Excel.Workbook(File.Contents("C:\data.xlsx"), null, true)`
- Create a table from a SQL Server database: `Table = Sql.Database("server", "database", [Query="SELECT * FROM table"])`
- Create a table from a web API: `Table = Json.Document(Web.Contents("https://api.example.com/data"))`

2. Table Transformation

- Rename a column: `Table.RenameColumns(Table, {"OldName", "NewName"})`
- Remove a column: `Table.RemoveColumns(Table, {"ColumnName"})`
- Filter rows based on a condition: `Table.SelectRows(Table, each [ColumnName] > 10)`
- Sort a table by a column: `Table.Sort(Table, {{"ColumnName", Order.Ascending}})`
- Group rows by a column and aggregate: `Table.Group(Table, {"GroupColumn"}, {{"AggregatedColumn", each List.Sum([ColumnToAggregate]), type number}})`
- Merge queries: `Table.NestedJoin(Table1, {"KeyColumn"}, Table2, {"ForeignKeyColumn"}, "NewColumnName", JoinKind.LeftOuter)`
- Append queries: `Table.Combine({Table1, Table2})`
- Pivot data: `Table.Pivot(Table, List.Distinct(Table[PivotColumn]), "PivotColumn", "ValueColumn", "AggregateFunction")`
- Unpivot data: `Table.UnpivotOtherColumns(Table, {"KeepColumn"}, "AttributeColumn", "ValueColumn")`
- Split a column by delimiter: `Table.SplitColumn(Table, "ColumnName", Splitter.SplitTextByDelimiter(","), {"Column1", "Column2"})`
- Combine columns: `Table.CombineColumns(Table, {"Column1", "Column2"}, Combiner.CombineTextByDelimiter(" ", QuoteStyle.None), "CombinedColumn")`
- Replace values: `Table.ReplaceValue(Table, "OldValue", "NewValue", Replacer.ReplaceText, {"ColumnName"})`
- Conditional column: `Table.AddColumn(Table, "NewColumn", each if [Condition] then "Value1" else "Value2")`

- Index column: `Table.AddIndexColumn(Table, "IndexColumn", 1, 1)`
- Duplicate column: `Table.DuplicateColumn(Table, "ColumnName", "NewColumnName")`

3. Data Cleansing

- Remove duplicates: `Table.Distinct(Table)`
- Remove nulls: `Table.SelectRows(Table, each not List.Contains(Record.FieldValues(_), null))`
- Fill down missing values: `Table.FillDown(Table, {"ColumnName"})`
- Fill up missing values: `Table.FillUp(Table, {"ColumnName"})`
- Replace errors: `Table.ReplaceErrorValues(Table, {{"ColumnName", "DefaultValue"}})`
- Trim whitespace: `Table.TransformColumns(Table, {{"ColumnName", Text.Trim, type text}})`
- Lowercase text: `Table.TransformColumns(Table, {{"ColumnName", Text.Lower, type text}})`
- Uppercase text: `Table.TransformColumns(Table, {{"ColumnName", Text.Upper, type text}})`
- Remove non-numeric characters: `Table.TransformColumns(Table, {{"ColumnName", each Text.Select(_, {"0".."9"}), type text}})`
- Remove non-alphabetic characters: `Table.TransformColumns(Table, {{"ColumnName", each Text.Select(_, {"a".."z", "A".."Z", " "}), type text}})`

4. Date and Time Operations

- Extract year from a date column: `Table.TransformColumns(Table, {{"DateColumn", Date.Year, Int64.Type}})`
- Extract month from a date column: `Table.TransformColumns(Table, {{"DateColumn", Date.Month, Int64.Type}})`
- Extract day from a date column: `Table.TransformColumns(Table, {{"DateColumn", Date.Day, Int64.Type}})`
- Extract hour from a time column: `Table.TransformColumns(Table, {{"TimeColumn", Time.Hour, Int64.Type}})`
- Extract minute from a time column: `Table.TransformColumns(Table, {{"TimeColumn", Time.Minute, Int64.Type}})`
- Extract second from a time column: `Table.TransformColumns(Table, {{"TimeColumn", Time.Second, Int64.Type}})`
- Calculate the difference between two dates: `Table.AddColumn(Table, "DaysDifference", each Duration.Days([EndDate] - [StartDate]))`

- Calculate the difference between two times: `Table.AddColumn(Table, "MinutesDifference", each Duration.TotalMinutes([EndTime] - [StartTime]))`
- Create a date column from year, month, and day columns: `Table.AddColumn(Table, "DateColumn", each #date([Year], [Month], [Day]))`
- Create a time column from hour, minute, and second columns: `Table.AddColumn(Table, "TimeColumn", each #time([Hour], [Minute], [Second]))`

5. Text Operations

- Concatenate columns: `Table.AddColumn(Table, "ConcatenatedColumn", each [Column1] & " " & [Column2])`
- Extract substring: `Table.TransformColumns(Table, {{"ColumnName", each Text.Range(_, 1, 5), type text}})`
- Find the position of a substring: `Table.TransformColumns(Table, {{"ColumnName", each Text.PositionOf(_, "Substring"), type number}})`
- Replace substring: `Table.TransformColumns(Table, {{"ColumnName", each Text.Replace(_, "OldSubstring", "NewSubstring"), type text}})`
- Split text by delimiter: `Table.TransformColumns(Table, {{"ColumnName", each Text.Split(_, ","), type text}})`
- Merge text with delimiter: `Table.TransformColumns(Table, {{"ColumnName", each Text.Combine(_, ";"), type text}})`
- Capitalize first letter: `Table.TransformColumns(Table, {{"ColumnName", each Text.Proper(_), type text}})`
- Reverse text: `Table.TransformColumns(Table, {{"ColumnName", each Text.Reverse(_), type text}})`
- Count occurrences of a substring: `Table.TransformColumns(Table, {{"ColumnName", each List.Count(Text.PositionOfAny(_, {"Substring"}, Occurrence.All)), type number}})`
- Pad text with characters: `Table.TransformColumns(Table, {{"ColumnName", each Text.PadStart(_, 10, "0"), type text}})`

6. Numeric Operations

- Round numbers: `Table.TransformColumns(Table, {{"ColumnName", each Number.Round(_, 2), type number}})`
- Absolute value: `Table.TransformColumns(Table, {{"ColumnName", each Number.Abs(_), type number}})`
- Square root: `Table.TransformColumns(Table, {{"ColumnName", each Number.Sqrt(_), type number}})`

- Logarithm: `Table.TransformColumns(Table, {{"ColumnName", each Number.Log(_), type number}})`
- Exponential: `Table.TransformColumns(Table, {{"ColumnName", each Number.Exp(_), type number}})`
- Trigonometric functions (sin, cos, tan): `Table.TransformColumns(Table, {{"ColumnName", each Number.Sin(_), type number}})`
- Floor and ceiling: `Table.TransformColumns(Table, {{"ColumnName", each Number.Floor(_), type number}})`
- Calculate the sum of a column: `List.Sum(Table[ColumnName])`
- Calculate the average of a column: `List.Average(Table[ColumnName])`
- Calculate the minimum and maximum of a column: `List.Min(Table[ColumnName]), List.Max(Table[ColumnName])`

7. Conditional Operations

- Conditional column based on a single condition: `Table.AddColumn(Table, "NewColumn", each if [Condition] then "Value1" else "Value2")`
- Conditional column based on multiple conditions: `Table.AddColumn(Table, "NewColumn", each if [Condition1] then "Value1" else if [Condition2] then "Value2" else "Value3")`
- Conditional column based on a nested if-then-else: `Table.AddColumn(Table, "NewColumn", each if [Condition1] then if [Condition2] then "Value1" else "Value2" else "Value3")`
- Conditional column based on a list of values: `Table.AddColumn(Table, "NewColumn", each if List.Contains({"Value1", "Value2"}, [ColumnName]) then "Match" else "No Match")`
- Conditional column based on a pattern: `Table.AddColumn(Table, "NewColumn", each if Text.StartsWith([ColumnName], "Prefix") then "Match" else "No Match")`

8. Aggregation and Grouping

- Group by a single column and count rows: `Table.Group(Table, {"GroupColumn"}, {{"Count", each Table.RowCount(_), type number}})`
- Group by multiple columns and sum a column: `Table.Group(Table, {"GroupColumn1", "GroupColumn2"}, {{"Sum", each List.Sum([ColumnToSum]), type number}})`
- Group by a column and calculate multiple aggregations: `Table.Group(Table, {"GroupColumn"}, {{"Sum", each List.Sum([ColumnToSum]), type number}, {"Average", each List.Average([ColumnToAverage]), type number}})`

- Group by a column and calculate a custom aggregation: `Table.Group(Table, {"GroupColumn"}, {{"Custom", each Text.Combine([ColumnToCombine], ","), type text}})`
- Group by a column and calculate a conditional aggregation: `Table.Group(Table, {"GroupColumn"}, {{"ConditionalSum", each List.Sum(Table.SelectRows(_, each [Condition])[ColumnToSum]), type number}})`

9. Joining and Merging

- Inner join two tables: `Table.Join(Table1, "JoinColumn", Table2, "JoinColumn", JoinKind.Inner)`
- Left outer join two tables: `Table.Join(Table1, "JoinColumn", Table2, "JoinColumn", JoinKind.LeftOuter)`
- Right outer join two tables: `Table.Join(Table1, "JoinColumn", Table2, "JoinColumn", JoinKind.RightOuter)`
- Full outer join two tables: `Table.Join(Table1, "JoinColumn", Table2, "JoinColumn", JoinKind.FullOuter)`
- Cross join two tables: `Table.CrossJoin(Table1, Table2)`
- Merge multiple tables vertically: `Table.Combine({Table1, Table2, Table3})`
- Merge multiple tables horizontally: `Table.Join(Table1, "JoinColumn", Table2, "JoinColumn", JoinKind.LeftOuter, MissingField.Ignore)`

10. Data Validation

- Check if a column contains only numeric values: `Table.AddColumn(Table, "IsNumeric", each List.AllTrue(List.Transform(Table[ColumnName], each Value.Is(_, type number))))`
- Check if a column contains only text values: `Table.AddColumn(Table, "IsText", each List.AllTrue(List.Transform(Table[ColumnName], each Value.Is(_, type text))))`
- Check if a column contains only dates: `Table.AddColumn(Table, "IsDate", each List.AllTrue(List.Transform(Table[ColumnName], each Value.Is(_, type date))))`
- Check if a column contains only values from a list: `Table.AddColumn(Table, "IsValidValue", each List.AllTrue(List.Transform(Table[ColumnName], each List.Contains({"Value1", "Value2"}, _))))`
- Check if a column contains only unique values: `Table.AddColumn(Table, "IsUnique", each List.Count(List.Distinct(Table[ColumnName])) = Table.RowCount(Table))`

- Check if a column contains any null values: `Table.AddColumn(Table, "HasNulls", each List.AnyTrue(List.Transform(Table[ColumnName], each _ = null)))`
- Check if a column matches a pattern: `Table.AddColumn(Table, "MatchesPattern", each List.AllTrue(List.Transform(Table[ColumnName], each Text.Contains(_, "Pattern"))))`
- Check if a column falls within a range: `Table.AddColumn(Table, "IsInRange", each List.AllTrue(List.Transform(Table[ColumnName], each _ >= Min and _ <= Max)))`

11. Data Profiling

- Calculate the count of rows: `Table.RowCount(Table)`
- Calculate the count of columns: `Table.ColumnCount(Table)`
- Get the list of column names: `Table.ColumnNames(Table)`
- Get the data types of columns: `Table.Schema(Table)[Name]`
- Calculate the distinct count of values in a column: `Table.AddColumn(Table, "DistinctCount", each List.Count(List.Distinct(Table[ColumnName])))`
- Calculate the percentage of null values in a column: `Table.AddColumn(Table, "NullPercentage", each List.Count(List.FindText(Table[ColumnName], null)) / Table.RowCount(Table))`
- Calculate the minimum and maximum values in a column: `Table.AddColumn(Table, "MinValue", each List.Min(Table[ColumnName])), Table.AddColumn(Table, "MaxValue", each List.Max(Table[ColumnName]))`
- Calculate the average and standard deviation of a column: `Table.AddColumn(Table, "Average", each List.Average(Table[ColumnName])), Table.AddColumn(Table, "StandardDeviation", each List.StandardDeviation(Table[ColumnName]))`
- Calculate the frequency distribution of values in a column: `Table.Group(Table, {"ColumnName"}, {{"Frequency", each Table.RowCount(_), type number}})`
- Calculate the length distribution of values in a column: `Table.AddColumn(Table, "Length", each Text.Length([ColumnName])), Table.Group(Table, {"Length"}, {{"Frequency", each Table.RowCount(_), type number}})`

12. Advanced Transformations

- Pivot data by multiple columns: `Table.Pivot(Table, List.Distinct(Table[PivotColumn1]), "PivotColumn1", List.Distinct(Table[PivotColumn2]), "PivotColumn2", "ValueColumn", List.Sum)`
- Unpivot data by multiple columns: `Table.UnpivotOtherColumns(Table, {"KeepColumn1", "KeepColumn2"}, "AttributeColumn", "ValueColumn")`
- Transpose a table: `Table.Transpose(Table)`
- Create a running total column: `Table.AddColumn(Table, "RunningTotal", each List.Sum(Table.Column(Table, "ValueColumn"){0..Table.RowCount(Table)-1}))`
- Create a window function: `Table.AddColumn(Table, "MovingAverage", each List.Average(Table.Column(Table, "ValueColumn"){Max(0, [RowNumber]-3)..Min(Table.RowCount(Table)-1, [RowNumber]+3)}))`

13. Error Handling

- Replace errors with a default value: `Table.ReplaceErrorValues(Table, {{"ColumnName", "DefaultValue"}})`
- Filter out rows with errors: `Table.SelectRows(Table, each not Table.HasErrors(_))`
- Identify rows with errors: `Table.AddColumn(Table, "HasErrors", each Table.HasErrors(_))`
- Capture errors into a new column: `Table.TransformColumns(Table, {{"ColumnName", each try _ otherwise null}})`

14. Performance Optimization

- Use query folding for efficient data retrieval: `Table.Buffer(Table)`
- Avoid using custom functions in large datasets: `Table.TransformColumns(Table, {{"ColumnName", each if _ > 10 then "High" else "Low"}})`
- Use query parameters for dynamic filtering: `Table.SelectRows(Table, each [ColumnName] = Parameter1)`
- Materialized calculated columns for frequently used calculations: `Table.AddColumn(Table, "CalculatedColumn", each [Column1] + [Column2], {"CalculatedColumn"})`
- Use indexing for faster lookups: `Table.AddIndexColumn(Table, "IndexColumn", 0, 1)`

15. Data Security

- Apply row-level security: `Table.SelectRows(Table, each [Username] = UsernameParameter)`
- Implement dynamic data masking: `Table.TransformColumns(Table, {{"ColumnName", each if not [IsSensitive] then _ else "********"}})`
- Use data source permissions for secure access: `Sql.Database("server", "database", [Query="SELECT * FROM table WHERE Username = '" & UsernameParameter & "'"])`

16. Data Hierarchies

- Create a parent-child hierarchy: `Table.TransformColumns(Table, {{"ParentColumn", each Table.SelectRows(Table, each [ChildColumn] = _)[ParentColumn]{0}}})`
- Create a level-based hierarchy: `Table.AddColumn(Table, "Level", each if [ParentColumn] = null then 0 else Table.SelectRows(Table, each [ChildColumn] = [ParentColumn])[Level]{0} + 1)`
- Aggregate data at different hierarchy levels: `Table.Group(Table, {"Level"}, {{"AggregatedValue", each List.Sum([Value]), type number}})`

17. Data Lineage

- Track data transformations using annotations: `Table.TransformColumns(Table, {{"ColumnName", each _, type text, Annotation.Combine(Annotation.Create("Source", "OriginalColumnName"))}})`
- Document data sources and transformations: `Sql.Database("server", "database", [Query="SELECT * FROM table", Documentation="Data retrieved from SQL Server"])`
- Use query groups to organize related queries: `Query.ApplyGroups(QueryGroup, {"Group1", "Group2"})`

18. Data Storytelling

- Create calculated measures for key performance indicators (KPIs): `Measure = CALCULATE(SUM(Table[ColumnName]), Table[FilterColumn] = "FilterValue")`
- Use bookmarks to create interactive data stories: `Bookmarks.Add("BookmarkName", Visuals.Snapshot)`
- Apply conditional formatting to highlight important insights: `Table.TransformColumns(Table, {{"ColumnName", each _, type text, Conditional.Format(Conditional.GreaterThan(10), "Green")}})`